

DOCUMENT RESUME

ED 084 781

EM 011 530

AUTHOR Gomberg, Sara
 TITLE JCL Procedures for Use of the Batch REL System.
 Revised Report.
 INSTITUTION California Inst. of Tech., Pasadena.
 SPONS AGENCY National Science Foundation, Washington, D.C.; Office
 of Naval Research, Washington, D.C.; Rome Air
 Development Center, Griffiss AFB, N.Y.
 REPORT NO CIT-REL-R-2
 PUB DATE Feb 73
 NOTE 27p.; See Also EM 011 531 through EM 011 536
 EDRS PRICE MF-\$0.65 HC-\$3.29
 DESCRIPTORS *Computers; Computer Science; *Interaction; *Man
 Machine Systems; Programing; *Programing Languages;
 Technical Reports; Time Sharing
 IDENTIFIERS Batch Processing; IBM 2314 Discs; IBM 3330 Discs;
 JCL; *Job Control Language; OS 360; OS 370; Rapidly
 Extensible Language; REL

ABSTRACT

Nine job control language (JCL) procedures for the Rapidly Extensible Language (REL) batch system are presented. The REL system itself is a total software system designed to facilitate communication with the computer in very high level languages. Its goal is an interactive time-shared system, but for a variety of reasons it has been desirable to develop an REL batch processing system. This batch system is designed to be used in any OS/360 or OS/370 batch environment with certain minimal hardware facilities, essentially, with run time core partition of 100k bytes or greater and standard OS interface to IBM 2314 or 3330 discs. The procedures are presented in three groups according to the following classes of users: 1) general consumers of REL languages, 2) language implementers and programing users, and 3) REL system programers. Brief descriptions of the data sets used are included. (Author)

FILMED FROM BEST AVAILABLE COPY

a project report on

REL

Co-principal investigators :

Bozena Henisz Dostert

Frederick B. Thompson

California Institute of Technology
Pasadena, California, 91109

EW 011 5-30

ED 084781

ED 084781

U S DEPARTMENT OF HEALTH,
EDUCATION & WELFARE
NATIONAL INSTITUTE OF
EDUCATION

THIS DOCUMENT HAS BEEN REPRODUCED EXACTLY AS RECEIVED FROM THE PERSON OR ORGANIZATION ORIGINATING IT. POINTS OF VIEW OR OPINIONS STATED DO NOT NECESSARILY REPRESENT OFFICIAL NATIONAL INSTITUTE OF EDUCATION POSITION OR POLICY.

JCL Procedures for Use of
the BATCH REL SYSTEM

Sara Gomborg

REL Report No. 2
(revised)

California Institute of Technology
Pasadena, California
February, 1973

The research reported here was supported in part
by the grants:

National Science Foundation grant# GH-31573
Rome Air Development Center contract# F30602-72-C-0249
Office of Naval Research contract# N00014-67-A-0094-0024

INTRODUCTION

The REL system is a total software system designed to facilitate communication with the computer in very high level languages. The goal of the REL project is an interactive, time shared system. However, for a variety of reasons, it has been desirable to develop an REL batch processing system. This REL batch system is designed to be used in any CS/360 or OS/370 batch environment with certain minimal hardware facilities, essentially, with run time core partition of 100K bytes or greater and standard CS interface to IBM 2314 or 3330 disks.

To operate in this OS batch environment, job control must be provided. This paper presents the nine JCL procedures for using the REL batch system.

The procedures are presented in three groups according to classes of users: 1. general consumers of REL languages, 2. language implementers and programming users, and 3. REL system programmers. A brief description of the data sets used is included.

PROCEDURES FOR GENERAL CONSUMERS OF REL LANGUAGES

BREL is the "go" step. It is the main driver procedure for the REL system. It accepts input sentences in an PEL language and processes them. If the language to be utilized at the time a call to BREL is made is not in existence, BREL can be used to put the language into the system (more about this is in the later section for language implementors).

BREL has no parameters. * To process input sentences you need to execute the procedure by the statements:

```
//          EXEC          BREL
//REL.INPUT DD          *
```

Then you need to give your authorized initials issued to you by your data base manager. This will insure that you are a valid user.

The format is determined by your installation.

Eg.
FBT

Once you have identified yourself to the system you are automatically entered into the command language. At this point, you have several command language statements from which to choose. Most likely you will want to enter your own language so that you can work on your data. (You may also want to alter, copy, create or delete your language.) This is done by:

* There is a parameter for use only in the Caltech environment:
DUMP = <dump name> (<dump name> is any 8 character or less name to identify an abend dump on disk which will be used for later debug on 2250 system.)

ENTER <language name>

Input statements

Each language has its own exit terminology. To leave the system you need a card containing:

EXIT

An example of the whole procedure is:

```
//      EXEC      BREL
//REL.INPUT DD      *
FBT
ENTER TONGA      (Tonga is a version of English)
HOW MANY WIVES OF LANDOWNERS HAVE MORE THAN 10 CHILDREN?
EXIT      (The exit terminology for the language Tonga)
ENTER AFL
DEF: BOX := ((-1, -1) → (1, 1), 1, 1, 90*V) FOR V=0, 3
      (see AFL paper for further description)
ESCAPE      (The exit terminology for AFL)
EXIT      (exit from the command language)
//
```

PROCEDURES FOR LANGUAGE IMPLEMENTORS

An REL language has two components. One part resides in the REL system and consists of the syntactic description and processing routines of the language. Since the REL system, at this time, does not contain facilities for programming these routines, the second component of a language is a set of aids for doing the programming in OS/370.

The routines must be written in assembly language and the following procedures are designed to aid in their manipulation in the OS environment. Each REL language has associated with it two OS partitioned data sets, a macro library and an object module library. The procedures create and delete these data sets; add a routine or macro to the appropriate library; and compress the libraries.

RELINIT

RELINIT creates the OS/370 data sets for a new language. They are <lang>.OBJLIB which contains object decks output by the assembler and <lang>.MACLIB which contains the macros to be used during assembly.

RELINIT has three parameters: 'LANG=', 'MACTRK=' and 'OBJTRK='. Each language has a name, 8 characters or less, written 'LANG=<name>', which should be chosen with care, so as not to conflict with another user's language.

The name REL1 may not be given to any language. 'MACTRK=n' is the parameter that allocates the number of tracks for the macro library. A general rule of thumb (on 2314's) is to estimate 70 cards per track + 5-10% extra. 'OBJTRK=m' is the parameter that allocates the number of tracks for the object module library. Ten tracks is sufficient for a small language. English uses ~50 tracks.

RELASM

RELASM is an assembly procedure. It assembles a source program and places it in the object library specified by the parameter 'LANG='. All assembly listings and error messages are printed.

Three macro libraries are referenced: <lang>.MACLIB, REL1.MACLIB, and SYS1.MACLIB in this order. These macro libraries can be overridden for special purposes (see the OS/370 JCL User's Guide for information on overriding catalogued procedures).

There are two parameters required by RELASM. The first 'LANG=<lang>' gives the name of the language under which the deck is to be assembled. 'NAME=<member name>' gives the name of the routine or deck that is being assembled. The following procedure would assemble a given deck into the object library of the language specified:

```
// EXEC RELASM, LANG=<lang>, NAME=<member name>
//ASM. SYSIN DD *
```

source deck

```
/*
//
```

RELDEL

This procedure deletes any data sets associated with the parameter 'LANG='. For example, suppose that the language 'OLD' is no longer needed. The following call would remove all 'OLD' data sets from the system.

```
// EXEC RELDEL, LANG=CLD
```

Ah, but be careful when using this procedure that an active language is not deleted.

RELCOMP

This routine compresses the language data sets. This operation is particularly useful when debugging routines. When an object module or a macro is replaced in a library, the disk space taken by the old version is not made available for reallocation until the library is compressed. Thus, if a library is used heavily for debugging, quite often it will run out of disk space. When this occurs RELCOMP should be called to reclaim any unused space. Suppose the English object or macro library were to run out of disk space, the following would reclaim any unused space:

```
// EXEC RELCOMP, LANG=ENGLISH
```

RELUPDTE

This procedure updates the macro library by putting the desired macros in the macro library of the language given by the parameter 'LANG='. The following sequence would add or replace macros in the desired macro library.

```
//          EXEC  RELUPDTE, LANG=<name>
//REL.INPUT DD      *
```

```
./ ADD LIST=ALL, NAME=<macro name>
```

```
    macro definition deck
```

```
./ ADD LIST=ALL, NAME=<macro name>
```

```
    macro definition deck
```

Note that every macro definition deck is preceded by a card:

```
./ ADD LIST=ALL, NAME=<macro name>
```

See the OS/ASSEMBLER book for a description of how to write macros.

Creating an REL Language

In creating a language you need to give the language a syntax and to put the semantic and syntactic routines in a form that is usable by the language. The following is an example:

```

//          EXEC      BREL
//REL.INPUT DD      *
FBT
CREATE VN1
ENTER VN1
METALANGUAGE
    {syntax
    {statements}
$ALTINPUT ROU
    | object decks of |
    | routines        |
$ENDALT
LINK EDIT ROU
EXIT
EXIT
/*

```

In this example we have created a language VN1 with its syntax and routines. The sequence:

```

$ALTINPUT ROU
    | object deck of |
    | routines        |
$ENDALT

```

creates and names an "alternate input" data block, ROU, consisting of the object decks of routines. These routines are put into the version VN1 by the statement LINK EDIT ROU, and are cross referenced with the syntax.

Rather than being involved with card decks there is another way to create an alternate input block. The previous example would look as follows:

```

//STEP1      EXEC      SEQ, LANG=<name>
//SYSIN      DD        *
      | Routine names |
      | (one per card) |
      |-----|
/*
//STEP2      EXEC      BREL
//RELINPUT  DD        *
FBT
BASE VNi ON NOTHING
ENTER VNi
METALANGUAGE
$ALTINPUT ROU
//          DD        DSN=SEQ, DISP=(OLD, DELETE)
//          DD        *
$ENDALT
      {syntax
      {statements}
LINK EDIT ROU
EXIT
EXIT
//

```

The first procedure, STEP1, takes the routines, listed on the cards following the //SYSIN DD * statement, from the partitioned library and puts them into a sequential data set named SEQ. Then the routines are in a form that can be logically concatenated with the REL.INPUT data set. The concatenation is performed by the JCL statement:

```
//          DD          DSN=SEQ, DISP=(OLD, DELETE)
```

Note that the "alternate input" block must be placed before its use, e. g. in the LINK EDIT statement.

REL SYSTEMS PROGRAMMERS' PROCEDURES

SYSASM

This procedure assembles a system deck into REL1. OBJLIB, the system object library. This is the sequence:

```
//          EXEC  SYSASM, NAME=<name>
//ASM. SYSIN DD  *
```

```
      system
      assembly deck
```

```
/*
```

where <name> is the system deck name.

RELSYSGN

This procedure formats, initializes, and adds system utilities to the REL system. RELSYSGN also creates and initializes the command language and therefore clears the system of all versions. The procedure takes the non-resident system utilities designated, and link-edits them into the REL system. The following is the sequence of procedures.

```
//STEP1      EXEC  SEQ, LANG=REL1
//SYSIN      DD      *
UDSR TN                                           (must be first one)
<system utility name>
:
<system utility name>
/*
//STEP2      EXEC  RELSYSGN
//RELSYSGN. RELINPUT DD  DSN=SEQ, DISP=(OLD, DELETE)
```

LINKGEN

LINKGEN adds/replaces non-resident system utilities into the REL system without affecting any user versions or the command language. This routine is useful if you have already used RELSYSGN but find that you want to add or replace one of the routines. Rather than go through a complete RELSYSGN, you can use the similar, but time saving and less user-disturbing LINKGEN. The following is the sequence to accomplish this.

```
//STEP1          EXEC      SEQ, LANG=REL1
//SYSIN          DD        *
<system utility name>
<system utility name>
<system utility name>
/*
//STEP2          EXEC      LINKGEN
//LINKGEN.RELINPUT DD      DSN=//SEQ, DISP=(OLD, DELETE)
```

Both RELSYSGN and LINKGEN use the same program, SYSGEN, which has three parameters:

parameter 1.

{ SYSGEN
LINKGEN }

SYSGEN causes the system data sets and the command language to be cleared, while LINKGEN does neither.

parameter 2.

{ PRINT
OPCONS }

PRINT sends system messages to the printer, while OPCONS sends them to the system operator's console.

parameter 3.

{ FORMAT
NOFMT }

FORMAT writes blocks to fill the system data sets and initialize them (this takes some time). NOFMT assumes that this has already been done and skips the I/O. The system data sets must be formatted before use.

The default is PARM=(SYSGEN, PRINT, FORMAT)

DATA SETS

System Data Sets

The REL system utilizes several data sets. REL.SD001, REL.SD002... (number determined by installation) are the REL system data sets. They contain all the versions, user data, REL languages, etc., and are manipulated by any execution within the REL system. These data sets have the following characteristics:

RECFM=F, DSORG=DA, BLKSIZE=PAGESIZE.

The REL system page size is currently set at 2048 bytes. REL.OBJLIB contains all system and metalanguage object modules. REL1.MACLIB is the system macro library and is available to all programs assembled under the RELASM and SYSASM procedures. REL1.LINKTXT contains linkage editor control statements providing for the editing of load modules. This library is not directly available to users. REL1.LINKLIB contains load modules ready for direct execution.

Language Data Sets

Each REL language requires two partitioned data sets; <lang>.OBJLIB, <lang>.MACLIB, where <lang> represents the name of the language.

<lang>.MACLIB is the macro library for the language. <lang>.MACLIB, REL1.MACLIB and SYS1.MACLIB are referenced in this order, as discussed in the section of this report on RELASM.

<lang>. OBJLIB is the object module library for the language called <lang>. A language object module library contains syntactic condition routines, semantic routines, and language dependent routines, such as initializers and debugging routines.

The member name assigned to an object module provides the sole means of referencing that deck (e. g. when the deck is to be link-edited into an REL version).

APPENDIX A
LISTING OF THE PROCEDURES

1. BREL
2. RELINIT
3. RELASM
4. RELDEL
5. RELCOMP
6. RELUPDTE
7. SYSASM
8. RELSYSCN
9. LINKGEN
10. SEQ

MEMBER NAME	BREL	
//BREL	PROC DUMP=DELETEME,SYS=SD	00000000
//DELETEDP	EXEC PGM=CUTY0000,PARM='&DUMP'	00000010
//REL	EXEC PGM=BRELSYS	00000020
//RELPRINT	DD SYSOUT=A,DCB=BUFNO=1	00000030
//RELDUMP	DD DSN=DUMP1.&DUMP,UNIT=SYSDA,DISP=(NEW,DELETE,CATLG),	*00000040
//	SPACE=(1024,(180),RLSE),DCB=(RECFM=F,BLKSIZE=1024),	*00000050
//	VOLUME=SER=REL003	00000060
//RELS001	DD DSN=REL.&SYS.001,DISP=(OLD,KEEP,KEEP)	00000070
//RELS002	DD DSN=REL.&SYS.002,DISP=(OLD,KEEP,KEEP)	00000080
//RELS003	DD DSN=REL.&SYS.003,DISP=(OLD,KEEP,KEEP)	00000090
//RELOUTPT	DD SYSOUT=A,UNIT=(SYSDA,SEP=(RELS003))	00000100
//RELINPUT	DD DDNAME=INPUT	00000110

MEMBER NAME RELINIT

//RELINIT PROC LANG=,MACTRK=5,OBJTRK=10

//CREATE EXEC PGM=IEFBR14

//MACLIB DD

//
// DSN=&LANG..MACLIB,DISP=(NEW,CATLG),UNIT=SYSDA,
SPACE=(TRK,(&MACTRK,,5),,CONTIG),VOL=SER=RELSYS,
DCB=(RECFM=FB,LRECL=80,BLKSIZE=1680)

//OBJLIB DD

// DSN=&LANG..OBJLIB,DISP=(NEW,CATLG),UNIT=SYSDA,
// SPACE=(TRK,(&OBJTRK,,5),,CONTIG),VOL=SER=RELSYS,
// DCB=(RECFM=FB,LRECL=80,BLKSIZE=800)

00000010

00000020

*00000030

*00000040

00000050

*00000060

*00000070

00000080

MEMBER NAME RELASM

//RELASM	PROC	GUMOD=,LANG=,NAME=#DUMMY,LOAD=,DECK=NU	00000000
//ASM	EXEC	PGM=IEUASM,PARM='&LOAD.LOAD,&DECK.DECK,RENT'	00000010
//SYSLIB	DD	DSN=&LANG..MACLIB,DISP=(SHR,PASS)	00000020
//	DD	DSN=REL1.MACLIB,DISP=(SHR,PASS)	00000030
//	DD	DSNAME=SYS1.MACLIB,DISP=(SHR,PASS)	00000040
//SYSUT1	DD	UNIT=(2314,SEP=SYSLIB),SPACE=(1700,(400,50))	00000050
//SYSUT2	DD	UNIT=2314,SPACE=(1700,(400,50))	00000060
//SYSUT3	DD	UNIT=(2314,SEP=(SYSUT1,SYSUT2,SYSLIB)),	*00000070
//		SPACE=(1700,(400,50))	00000080
//SYSPRINT	DD	SYSOUT=A	00000090
//SYSGU	DD	&GOMOD.DSN=&LANG..OBJLIB(&NAME),DISP=(OLD,PASS),	*00000100
//		DCB=(RECFM=FB,LRECL=80,BLKSIZE=800)	00000110

MEMBER NAME RELDEL

//RELDEL PROC LANG=

//DELETE EXEC PGM=IEFBR14

//OBJLIB DD DSN=&LANG..OBJLIB,DISP=(OLD,DELETE)

//MACLIB DD DSN=&LANG..MACLIB,DISP=(OLD,DELETE)

00000000

00000010

00000020

00000030

MEMBER NAME RELCOMP
//RELCOMP PROC LANG=
//COMPOBJ EXEC PGM=IEBCOPY,PARM='COMPRESS'
//SYSPRINT DD SYSOUT=A
//SYSUT1 DD DSNAME=&LANG..OBJLIB,DISP=OLD
//SYSUT2 DD DSNAME=&LANG..OBJLIB,DISP=OLD
//SYSIN DD DUMMY
//COMPMAC EXEC PGM=IEBCOPY,PARM='COMPRESS'
//SYSPRINT DD SYSOUT=A
//SYSUT1 DD DSNAME=&LANG..MACLIB,DISP=OLD
//SYSUT2 DD DSNAME=&LANG..MACLIB,DISP=OLD
//SYSIN DD DUMMY

00000000
00000010
00000020
00000030
00000040
00000050
00000060
00000070
00000080
00000090
00000100

MEMBER NAME RELUPDTE
//RELUPDTE PROC LANG=
//REL EXEC PGM=IEBUPDTE,PARM=NEW
//SYSPRINT DD SYSOUT=A
//SYSUT2 DD DSN=&LANG..MACLIB,DISP=OLD
//SYSIN DD DDNAME=INPUT

00000000
00000010
00000020
00000030
00000040

MEMBER NAME		SYSASM	
//ASMLoad	PROC	LOAD=,GOMOD=,DECK=NO,NAME=#DUMMY,LIB=1	00000010
//ASM	EXEC	PGM=IEUASM,PARM=(&LOAD.LOAD,&DECK.DECK)	00000020
//SYSPRINT	DD	SYSOUT=A	00000030
//SYSLIB	DD	DSN=REL1.MACLIB,DISP=(SHR,PASS)	00000040
//	DD	DSN=SYS1.MACLIB,DISP=(SHR,PASS)	00000050
//SYSUT1	DD	SPACE=(1700,(400,50)),UNIT=(2314,SEP=SYSLIB)	00000060
//SYSUT2	DD	SPACE=(1700,(400,50)),UNIT=2314	00000070
//SYSUT3	DD	SPACE=(1700,(400,50)),	*00000080
//		UNIT=(2314,SEP=(SYSLIB,SYSUT1,SYSUT2))	00000090
//SYSGU	DD	DSN=&&OBJMOD,DISP=(,PASS),UNIT=SYSDA,	*00000100
//		SPACE=(80,(40,10))	00000110
//LOAD	EXEC	PGM=IEBGENER,COND=(4,LE,ASM)	00000120
//SYSPRINT	DD	DUMMY	00000130
//SYSIN	DD	DUMMY	00000140
//SYSUT1	DD	DSN=&&OBJMOD,DISP=(OLD,DELETE)	00000150
//SYSUT2	DD	&GOMOD.DSN=REL&LIB..OBJLIB(&NAME),DISP=(OLD,PASS),	*00000160
//		DCB=(RECFM=FB,BLKSIZE=800,LRECL=80)	00000170

MEMBER NAME	RELSYSGN	
//RELSYSGN	PROC DCLS=D,SYS=SD,LIB=1	00000010
//LKED	EXEC PGM=IEWL,PARM='LET,LIST,XREF,SIZE=(88K,8K)'	00000020
//SYSPRINT	DD SYSOUT=&DCLS	00000030
//SYSLIB	DD DSN=REL&LIB..OBJLIB,DISP=SHR	00000040
//	DD DSN=REL1.OBJLIB,DISP=SHR	00000050
//SYSUT1	DD UNIT=2314,SPACE=(1024,(50,20))	00000060
//SYSLMOD	DD DSN=&&LINKSET,DISP=(,PASS),UNIT=2314,	*00000070
//	SPACE=(1024,(100,20,3))	00000080
//SYSLIN	DD DSN=REL1.LINKTXT(RELGEN),DISP=SHR	00000090
//RELSYSGN	EXEC PGM=SYSGEN,COND=(8,LT,LKED),PARM=(SYSGEN,PRINT,NOFMT)	00000100
//STEPLIB	DD DSN=&&LINKSET,DISP=(OLD,DELETE,DELETE)	00000110
//RELOUTPT	DD SYSOUT=A	00000120
//RELPRINT	DD SYSOUT=A,DCB=BUFNO=1	00000130
//SYSUDUMP	DD SYSOUT=&DCLS	00000140
//RELS001	DD DSN=REL.&SYS.001,DISP=(OLD,PASS,KEEP)	00000150
//RELS002	DD DSN=REL.&SYS.002,DISP=(OLD,PASS,KEEP)	00000160
//RELS003	DD DSN=REL.&SYS.003,DISP=(OLD,PASS,KEEP)	00000170

MEMBER NAME LINKGEN		
//LINKGEN	PROC DCLS=A,SYS=SD,LIB=1	00000010
//LKED	EXEC PGM=IEWL,PARM='LET,LIST,XREF,SIZE=(88K,8K)'	00000020
//SYSPRINT	DD SYSOUT=A	00000030
//SYSLIB	DD DSN=REL&LIB..OBJLIB,DISP=SHR	00000040
//	DD DSN=REL1.OBJLIB,DISP=SHR	00000050
//SYSUT1	DD UNIT=2314,SPACE=(1024,(50,20))	00000060
//SYSLMOD	DD DSN=&&LINKSET,DISP=(,PASS),UNIT=2314,	*00000070
//	SPACE=(1024,(100,20,3))	00000080
//SYSLIN	DD DSN=REL1.LINKTXT(RELGEN),DISP=SHR	00000090
//LINKGEN	EXEC PGM=SYSGEN,COND=(8,LT,LKED),PARM=(LINKGEN,PRINT)	00000100
//STEPLIB	DD DSN=&&LINKSET,DISP=(OLD,DELETE,DELETE)	00000110
//RELOUTPT	DD SYSOUT=A	00000120
//RELPRINT	DD SYSOUT=A,DCB=BUFNO=1	00000130
//SYSUDUMP	DD SYSOUT=&DCLS	00000140
//RELS001	DD DSN=REL.&SYS.001,DISP=(OLD,PASS,KEEP)	00000150
//RELS002	DD DSN=REL.&SYS.002,DISP=(OLD,PASS,KEEP)	00000160
//RELS003	DD DSN=REL.&SYS.003,DISP=(OLD,PASS,KEEP)	00000170

MEMBER NAME SEQ

//SEQ	PROC	LANG=,DSN=	00000000
//SEQ	EXEC	PGM=SEQUENT	00000010
//SYSPRINT	DD	SYSOUT=A	00000020
//SYSLIB	DD	DSN=&LANG..OBJLIB,DISP=SHR	00000030
//SYSLOUT	DD	DSN=&&SEQ&DSN,DISP=(MOD,PASS),UNIT=SYSDA,	*00000040
//		SPACE=(TRK,(30,10)),	*00000050
//		DCB=(RECFM=FB,LRECL=80,BLKSIZE=800)	00000060